Title	Towards payment-bound analysis in cloud systems with task-prediction errors
Author(s)	Di, S; Wang, CL; Kondo, D; Han, G
Citation	The 6th International Conference on Cloud Computing (CLOUD 2013), Santa Clara Marriott, CA., 27 June-2 July 2013.
Issued Date	2013
URL	http://hdl.handle.net/10722/189639
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Towards Payment-Bound Analysis in Cloud Systems with Task-Prediction Errors

Sheng Di[1], Cho-Li Wang[2], Derrick Kondo[1], Guodong Han[2]
[1]INRIA, France, [2]The University of Hong Kong, Hong Kong
sheng.di@inria.fr, clwang@cs.hku.hk, derrick.kondo@inria.fr, gdhan@cs.hku.hk

***Abstract*—In modern cloud systems, how to optimize user service level based on virtual resources customized on demand is a critical issue. In this paper, we comprehensively analyze the payment bound under a cloud model with virtual machines (VMs), by taking into account that task's workload may be predicted with errors. The analysis is based on an optimized resource allocation algorithm with polynomial time complexity. We theoretically derive the upper bound of task payment based on a particular margin of workload prediction-error. We also extend the payment-minimization algorithm to adapt to the dynamic changes of host availability over time, and perform the evaluation by a real-cluster environment with 56 VMs deployed. Experiments confirm the correctness of our theoretical inference, and show that our payment-minimization solution can keep 95% of user payments below 1.15 times as large as the theoretical values of the ideal payment with hypothetically accurate information. The ratio for the rest user payments can be limited to about 1.5 at the worst case.**

I. Introduction

Cloud computing [1] has become a new effective service provisioning paradigm with the maximized flexibility under user's control. Rather than Grid computing, one key feature of Cloud computing is its elastic resource provisioning that allows users to customize resources on demand.

In our previous work [3], we built a Cloud resource allocation model that aims to minimize user payment by leveraging virtual machine (VM) technology. We proposed a polynomial algorithm, namely *MIN-Payment-OPT*, which can estimate the optimal resource allocation for tasks within their budget requirements. Its output has been proved to be optimal based on Karush-Kuhn-Tucker (KKT) conditions [4]. In this paper, we further investigate the effectiveness of users' payments as compared to their budgets, by answering two questions below.

- What is the bound of task payment under the Min-Payment-OPT algorithm, provided that task workloads are predicted with a certain margin of error?
- What are the experimental results about payments in a VM-supported cloud environment with workload prediction errors, under various levels of competitions?

Our work with consideration of prediction errors is different from the existing task scheduling research. Many existing works [5], [6] assume task workload is exactly pre-known, while the others [7], [8], [9] assume such information is completely unknown. If the task workload can be predicted accurately, the resource allocation issue will actually be simplified as either convex-optimization problem [10], [11] or combinatorial-optimization problem [12], [13]). However, this will definitely raise a significant gap between the research and practice. By contrast, if task workload is assumed to be completely unknown, the problem becomes as intractable as a black-box, which can only be coped with via heuristics or experiences. Due to the increasingly mature workload prediction research like [14], task's multi-dimensional workloads could already be predicted in advance, yet suffering a certain margin of errors.

In this paper, we first theoretically derive the bounds of user payments about their task executions, given a margin of workload prediction error. Such an analysis is significant in that payment is often one of the most important requirements in cloud systems. In fact, if the relationship between user's real payment and task workload prediction errors is studied comprehensively, user's cost can be restricted well and they will be more satisfied with their task executions.

There are at least two key challenges in the theoretical analysis about payment bound. On one hand, it is critical to build an ease-of-use (or feasible) model to suit most of the workload prediction methods. On the other hand, it is necessary to comprehensively study the resource allocation scheme, so as to explore a set of intrinsic rules for deducing the conjectured payment bounds.

In our experiment, we implement a cloud prototype with over 56 VMs deployed. We defined a useful metric, called *Payment Extension Ratio* (PER), to evaluate the user payment level, under different levels of resource competition. Experiments show that the Min-Payment-OPT algorithm can effectively keep 95% of user payments with erroneous prediction information below 1.15 times as large as the theoretical values estimated with hypothetically accurate information. Our experiments also show PER for the rest tasks can be limited to about 1.5 at the worst case.

The rest of the paper is organized as follows. We first formulate the payment bound problem in Section II and briefly introduce the Min-Payment-OPT algorithm in Section III. In Section IV, we theoretically analyze the upper bound of the user payment, by taking into account the possible task workload prediction errors. In Section V, we present the experimental results. We also extend the basic Min-Payment-OPT algorithm to an online version, that can adapt to the dynamically changing node's availability states. In Section VI, we discuss the related work. Finally, we conclude the

paper with a vision of the future work in Section VII.

II. Problem Formulation

A. Payment Minimization Model

In our cloud model, a cloud proxy (a.k.a., server) will receive and respond to user requests (or tasks) with customized requirements (or virtual machines). As a task is submitted, it will be attached to an isolated VM, whose multiple resources (e.g., CPU rate, I/O rate) will be optimally customized. Upon its completion, the computational result will be sent to its user and the resources will be released for new usage.

Suppose there are n computational nodes (a.k.a., execution nodes, denoted by p_i, where $1 \leq i \leq n$) in the resource pool. For any particular task with R execution dimensions (e.g., R types of resources like CPU rate and I/O bandwidth), we use Π to denote the whole set of dimensions and $\boldsymbol{c}(p_i)=(c_1(p_i)\ ,\ c_2(p_i)\ ,\cdots,\ c_R(p_i))^T$ as node p_i's capacity vector. For example, a node p_1's physical capacity vector could be denoted as $\boldsymbol{c}(p_1)$={CPU_rate=2.4Gflops,disk_IO_rate=1Gbps}.

Any task is denoted as t_i, where $1 \leq i \leq m$, and m refers to the total number of submitted tasks. Each task has a multi-dimensional *workload vector*, denoted by $\boldsymbol{l}(t_i)=(l_1(t_i),l_2(t_i),\cdots,\ l_R(t_i))^T$, which needs to be finished before the task's deadline. We denote the resource vector allocated to t_i as $\boldsymbol{r}(t_i)\ =\ (r_1(t_i),\ r_2(t_i),\ \cdots\ , r_R(t_i))^T$, where $r_k(t_i)$ (k=1,2,$\cdots$,R) refers to the resource fraction on the kth dimension isolated for the task's execution. Node p_i's availability vector (denoted $\boldsymbol{a}(p_i)$) along the multiple dimensions is calculated by $\boldsymbol{c}(p_j) - \sum_{t_i\ running\ on\ p_j} \boldsymbol{r}(t_i)$. In the above example about node p_1, if it is running two VMs that are allocated with half of the total physical resources, its availability vector $\boldsymbol{a}(p_1)$ is equal to {CPU_rate=1.2Gflops,disk_IO_rate=0.5Gbps}. For simplicity, we denote task t_i's execution time (a.k.a., execution length) as Equation (1) (an affine transformation of $\sum_{i=1}^{R} \frac{l_k}{r_k}$), where θ denotes a constant coefficient. Such a definition specifies a defacto broad set of applications each with multiple execution dimensions. One typical example is a single job with multiple sequentially interdependent tasks or some program with distinct execution phases each relying on independent compute resources (then $\theta = 1$).

$$T(t_i) = \theta \sum_{k=1}^{R} \frac{l_k(t_i)}{r_k(t_i)}, where\ \theta \in [\frac{\max(\frac{l_k}{r_k})}{\sum_{k=1}^{R} \frac{l_k}{r_k}}, 1] \quad (1)$$

For any cloud system, the resources provisioned are usually set with a price vector denoted as $\boldsymbol{b}(p_i)=(b_1(p_i),\ b_2(p_i),\cdots,\ b_R(p_i))^T$ along R dimensions. $b_k(p_i)$ ($1 \leq k \leq R$) denotes the per-time-unit price of the kth dimension on p_i. Each task t_i is set with a deadline (denoted $D(t_i)$) for its execution and the payment is expected to be minimized.

Any task will be executed on one VM with *user-reserved* resources and the payment is calculated based on the customized resource (a.k.a., pay-by-reserve policy). A real cloud system that uses this policy is Haizea [16]. Adopting such a pricing policy is due to the fact that the efficiency of many applications relies on multiple resources but it is non-trivial to precisely evaluate the exact amount of their consumption separately on individual resources. Moreover, quite a few users prefer to reserving resources for tolerating usage burst and guaranteeing their service levels.

Task's total payment will be calculated via Equation (2), where p_s refers to the task t_i's execution node. The mean price (i.e., $\frac{1}{R}\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}(t_i)$) will be used as the pricing unit over time, for computing user's payment. Such a design can be consistent with our pay-by-reserve model, and also prevent payment cost from being too high when their applications' execution cannot overlap at different dimensions.

$$P(\boldsymbol{r}(t_i)) = \frac{1}{R}\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}(t_i) \cdot T(t_i) \quad (2)$$

In the paper, we will omit some redundant notations t_i and p_i if thus would not cause ambiguity. For instance, $l_k(t_i)$, $\boldsymbol{r}(t_i)$, $b_k(p_i)$, $\boldsymbol{a}(p_i)$ and $D(t_i)$ may be substituted by l_k, $\boldsymbol{r}$, b_k, $\boldsymbol{a}$, and D respectively, in the following text.

The *payment minimization* problem could be summarized as the following convex optimization format: for any task t_i with its workload vector $\boldsymbol{l}(t_i)$, given a set of candidate execution nodes (p_s, s=1,2,$\cdots$,n), how to select p_s and split resource such that t_i's payment (i.e., Equation (2)) is minimized, subject to the constraints (3) and (4).

$$min\ \ P(\boldsymbol{r}(t_i))$$

$$s.t.\ \ T(t_i) \leq D(t_i) \quad (3)$$

$$\boldsymbol{r}(t_i) \preceq \boldsymbol{a}(p_s) \quad (4)$$

We proposed a polynomial algorithm - Algorithm 1 (a.k.a., Min-Payment-OPT) to solve the above problem, which has been proved optimal in our previous work [3].

B. Payment Bounding Problem

Although Algorithm 1's output has been proved optimal, such a result relies on a strong condition, i.e., task's workload vector should be predicted accurately. In order to adapt to erroneous workload prediction, we define the margin of workload prediction error as follows.

Definition 1: Suppose a task t_i's real workload vector is $\boldsymbol{l}(t_i)$, while its workload vector used by our algorithm is $\boldsymbol{l'}(t_i)$ subject to Inequality (5), where α and β are the lower bound and upper bound of the margin of the prediction error specified by users based on experiences or particular workload prediction methods such as [14], [17], [18].

$$\alpha \leq \frac{l'_k(t_i)}{l_k(t_i)} \leq \beta,\ k = 1, 2, \cdots R \quad (5)$$

We give an example to illustrate the above definition. Suppose a task t_i's real workloads are always in [0.125, 1], and the workload vector $\boldsymbol{l'}(t_i)$ used by Algorithm 1 will be set based on the task's historical execution records. Specifically, each element $l'_k(t_i)$ ($k = 1, 2, \cdots R$) will be

set to 0.25 if the corresponding true workload fluctuates in [0.125, 0.5] and set to 0.75 if the true workload ranges within (0.5, 1]. Then, we could get Inequality (6) below, since α=$\frac{0.125}{0.25}$=0.5 and β=$\frac{0.5}{0.25}$=2.

$$0.5 \leq \frac{l'_k(t_i)}{l_k(t_i)} \leq 2,\ k = 1, 2, \cdots R \tag{6}$$

Using the inaccurate prediction $\boldsymbol{l}'(t_i)$ to run the Algorithm 1, it is obvious that t_i's real payment may be skewed more or less from its theoretical value estimated with accurate workload $\boldsymbol{l}(t_i)$. Hence, a critical question is what the payment bound will get when using $\boldsymbol{l}'(t_i)$, compared to its ideal value calculated with the accurate workload $\boldsymbol{l}(t_i)$.

III. Basic Min-Payment-OPT Algorithm

In this section, we briefly introduce the Min-Payment-OPT in Algorithm 1. Co-Step(Γ,C) is a key function (defined in Definition 2) to get the convex-optimal solution with unbounded resource assumption. The whole algorithm repeatedly tune the convex-optimal solutions computed by Co-Step(Γ,C), until getting a viable solution compatible with the conditions (3) and (4).

Algorithm 1 Basic Min-Payment-OPT Algorithm

Input: deadline $D(t_i)$; **Output**: execution node p_s, optimal resource $\boldsymbol{r}^*(t_i)$

for (each candidate node p_s) **do**
  $\Gamma = \Pi$, $C = D(t_i)$, $\boldsymbol{r}^* = \varnothing$ (empty set);
  repeat
    $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$ = Co-Step(Γ,C); /*Compute optimal $\boldsymbol{r}$ on Γ*/
    $\Omega = \{d_k | d_k \in \Gamma \ \&\ r_k^{(*)}(t_i, p_s) > a_k(p_s)\}$; /*select elements violating constraint (4)*/
    $\Gamma = \Gamma \backslash \Omega$; /*$\Gamma$ takes away Ω*/
    $C = C - \theta \sum_{d_k \in \Omega} \frac{l_k}{a_k}$; /*Update C*/
    $\boldsymbol{r}^*(t_i, p_s) = \boldsymbol{r}^*(t_i, p_s) \cup \{r_k^* = a_k(p_s) \mid d_k \in \Omega\ \&\ a_k(p_s)$ is d_k's upper bound$\}$;
  until $(\Omega = \varnothing)$;
  $\boldsymbol{r}^*(t_i, p_s) = \boldsymbol{r}^*(t_i, p_s) \cup \boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$;
end for
Select the smallest $P(t_i)$ by traversing the candidate solution set;
Output the selected node p_s and resource allocation $\boldsymbol{r}^*(t_i, p_s)$;

Definition 2: For any task t_i, based on a subset $\Gamma(\subseteq\Pi)$, Co-Step(Γ, C) is defined as the convex optimal solution to minimizing $P(\boldsymbol{r}_\Gamma(t_i))$ subject to the single constraint (7). C denotes a constant (e.g., deadline) and $\boldsymbol{r}_\Gamma(t_i)$ denotes the amounts of resources in Γ gained by t_i, where d_j is the corresponding dimension of l_j and b_j.

$$T(t_i) = \theta \sum\nolimits_{d_j \in \Gamma} \frac{l_j}{r_j} \leq C \tag{7}$$

The output of Co-Step(Γ, C) is denoted as $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$ =$(r_1^{(*)}(t_i), r_2^{(*)}(t_i), \cdots, r_{|\Gamma|}^{(*)}(t_i))^T$. The value of $r_k^{(*)}(t_i)$, as shown below, has been proved optimal in our previous work [3].

$$r_k^{(*)}(t_i) = \left(\frac{\theta}{C} \sum\nolimits_{d_j \in \Gamma} \sqrt{l_j b_j}\right) \sqrt{\frac{l_k}{b_k}} \tag{8}$$

IV. Theoretical Analysis of Payment Bound

In this section, we will derive an upper bound with respect to the user payment for the case with unbounded resource capacity, and an approximated upper bound for the case with bounded resource capacity. We denote P_E^* as the payment output of Algorithm 1 with an inaccurate workload prediction and denote P_I^* as the ideal payment output performed by the algorithm with accurate workload vector. Their corresponding resource vectors are denoted as r_E^* and r_I^* respectively. Hence, our objective is to determine the upper bound of $\frac{P_E^*}{P_I^*}$, a.k.a., payment approximation ratio.

Theorem 1: **Given** a task t_i with a predefined deadline $D(t_i)$, a candidate execution node p_s with unbounded available resource vector and a bounded price vector $\boldsymbol{b}(p_s)$, and a skewed workload vector $\boldsymbol{l}'(t_i)$ subject to Inequality (5), **then** under the resource allocation $\boldsymbol{r}_E^{(*)}$, the bound of t_i's payment conforms to Inequality (9). (Note $\boldsymbol{r}^{(*)}$ is defined as the ideal optimal resource vector calculated based on unbounded resource capacity (as shown in Formula (8)), so $\boldsymbol{r}_I^{(*)}$ and $\boldsymbol{r}_E^{(*)}$ denote the values of $\boldsymbol{r}^{(*)}$ calculated under accurate and inaccurate workload prediction respectively)

$$P_E^{(*)}(t_i) \leq \sqrt{\frac{\beta}{\alpha}} \cdot P_I^{(*)}(t_i) \tag{9}$$

Proof:

$$P_I^{(*)} = \tfrac{1}{R}\left(\sum_{k=1}^{R} b_k r_{Ik}^{(*)}\right) \cdot \left(\theta \sum_{k=1}^{R} \tfrac{l_k}{r_{Ik}^{(*)}}\right)$$

$$= \tfrac{1}{R}\left(\sum_{k=1}^{R} b_k \left(\tfrac{\theta}{D} \sum_{i=1}^{R} \sqrt{l_i b_i}\right) \sqrt{\tfrac{l_k}{b_k}}\right) \cdot \left(\theta \sum_{k=1}^{R} \frac{l_k}{\left(\tfrac{\theta}{D} \sum_{i=1}^{R} \sqrt{l_i b_i}\right) \sqrt{\tfrac{l_k}{b_k}}}\right)$$

$$= \tfrac{1}{R} \cdot \tfrac{\theta}{D} \left(\sum_{i=1}^{R} \sqrt{l_i b_i}\right) \left(\sum_{k=1}^{R} \sqrt{l_k b_k}\right) \cdot D \cdot \frac{\sum_{k=1}^{R} \sqrt{l_k b_k}}{\sum_{i=1}^{R} \sqrt{l_i b_i}}$$

$$= \tfrac{1}{R}\theta \left(\sum_{i=1}^{R} \sqrt{l_i b_i}\right)^2$$

$$P_E^{(*)} = \tfrac{1}{R} \cdot \left(\sum_{k=1}^{R} b_k r_{Ek}^{(*)}\right) \cdot \left(\theta \sum_{k=1}^{R} \tfrac{l_k}{r_{Ek}^{(*)}}\right)$$

$$= \tfrac{1}{R}\left(\sum_{k=1}^{R} b_k \left(\tfrac{\theta}{D} \sum_{i=1}^{R} \sqrt{l'_i b_i}\right) \sqrt{\tfrac{l'_k}{b_k}}\right) \cdot \left(\theta \sum_{k=1}^{R} \frac{l_k}{\left(\tfrac{\theta}{D} \sum_{i=1}^{R} \sqrt{l'_i b_i}\right) \sqrt{\tfrac{l'_k}{b_k}}}\right)$$

$$= \tfrac{1}{R} \cdot \tfrac{\theta}{D} \left(\sum_{i=1}^{R} \sqrt{l'_i b_i}\right)^2 \cdot D \cdot \frac{\sum_{k=1}^{R} l_k \sqrt{\frac{b_k}{l'_k}}}{\sum_{i=1}^{R} \sqrt{l'_i b_i}}$$

$$= \tfrac{1}{R}\theta \left(\sum_{i=1}^{R} \sqrt{l'_i b_i}\right) \cdot \sum_{k=1}^{R} l_k \sqrt{\tfrac{b_k}{l'_k}}$$

$$\leq \tfrac{\theta}{R} \left(\sum_{i=1}^{R} \sqrt{\beta l_i b_i}\right) \cdot \sum_{k=1}^{R} l_k \sqrt{\tfrac{b_k}{\alpha l_k}}$$

$$= \sqrt{\tfrac{\beta}{\alpha}} \cdot \tfrac{\theta}{R} \left(\sum_{i=1}^{R} \sqrt{l_i b_i}\right)^2 = \sqrt{\tfrac{\beta}{\alpha}} P_I^{(*)}$$ ■

We could get the above conclusion with the assumption, unbounded resource capacity. However, under the situation with bounded resource capacities, we cannot derive such a neat conclusion as Inequality (9), so that the ratio of

inaccurate-workload based output to the accurate-workload based output is subject to a constant upper bound. In Theorem 2, we still successfully derive an approximation upper bound by considering bounded available resources, despite the upper bound being not only determined by a constant ratio, but also related to a supplement.

In order to build the relation between P_E^* and P_I^*, we introduce $P_E^{[*]}$ in Definition 3 which can be proved no greater than P_I^* later. Then, we just need to compare P_E^* and $P_E^{[*]}$ instead.

Definition 3: For task t_i, given a subset $\Gamma(\subseteq\Pi)$ where their corresponding resource shares are already designated, then $r^{[*]}(\Gamma)$ is defined as such a resource allocation which minimizes the $P(t_i)$ with unbounded resource capacities.

We give an example to further describe the above definition. Suppose there is a task with three dimensions to process, and we also know its workload $\boldsymbol{l}$=$(10Gfloatpoints, 10Gbit, 20Gbit)^T$, its deadline is set to 30, and the availability vector and price vector of the specific candidate node p_s are (*CPU_rate*=5*GFlops*, *IO_rate* = 10*Gbps*, *NetBandwidth*=2*Gbps*)^T and $(1\$ps, 1\$ps, 2\$ps)^T$ respectively. Based on our initial problem formulation, the objective is to find an optimal divisible-resource allocation (such as (*CPU* = 3*GFlops*, *IOSpeed* = 4*Gbps*, *NetBandwidth*=1*Gbps*)^T) such that the payment cost is minimized, and Γ=$\varnothing$ (empty set) in this situation.

Differently, Definition 3 takes into account the situation that $\Gamma\neq\Phi$. That is, assuming a few resource fractions are already designated by system, for example, (*CPU_rate*=2*GFlops*,*IO_rate*=3.5*Gbps*)^T, then Γ={*CPU_rate*, *IO_rate*}, and $r^{[*]}(\Gamma)$ is the resource allocation (such as (*CPU_rate*=2*GFlops*, *IO_rate*=3.5*Gbps*, *NetBandwidth*=1.5*Gbps*)^T) of minimizing $P^{[*]}(\Gamma)$ subject to the above information.

We denote $P^{[*]}(t_i)$ the task t_i's payment cost when using the resource allocation $r^{[*]}(\Gamma)$ onto the task. It is obvious that Inequality (10) must hold, and we can also derive the resource allocation vector as Equation (11) in Lemma 1 for any $d_k\notin\Gamma$. Note that for any $d_k\in\Gamma$, r_k's value is already specified priori.

$$P^{[*]}(t_i) \leq P^*(t_i) \tag{10}$$

$$r_{Ik}^{[*]} = \frac{\theta\sum_{d_i\notin\Gamma}\sqrt{l_i b_i}}{D-\theta\sum\limits_{d_i\in\Gamma}\frac{l_i}{r_i}}\sqrt{\frac{l_k}{b_k}} \tag{11}$$

Lemma 1: For task t_i, given a subset $\Gamma(\subseteq\Pi)$ where their corresponding resource shares are already designated, then $r^{[*]}(\Gamma)$ can be calculated using the Formula (11), where $r_{Ik}^{[*]}$ is referred to the kth element of $r^{[*]}(\Gamma)$.

Proof: Based on Equation (2), we could get Equation (12), under the assumption that Γ contains all the resource amounts that are already specified.

$$\begin{aligned} &P_I^{[*]}(\boldsymbol{r}(t_i)) = \tfrac{1}{R}\cdot T^{[*]}(t_i)\cdot(\sum_{d_k\in\Gamma} b_k r_k + \sum_{d_k\notin\Gamma} b_k r_{Ik}^{[*]}) \\ &where\ T^{[*]}(t_i) = \sum_{d_k\in\Gamma}\tfrac{l_k}{r_k} + \sum_{d_k\notin\Gamma}\tfrac{l_k}{r_{Ik}^{[*]}} \end{aligned} \tag{12}$$

According to the Definition 3, we know that $r_{Ik}^{[*]}$ ($\forall d_k\notin\Gamma$) is computed using convex optimization with unbounded resource capacities, thus, $T^{[*]}=D$. Then, we can get the Lagrangian function as Equation (13) for $P_I^{[*]}(t_i)$, where λ is its Lagrangian multiplier.

$$F_3(\boldsymbol{r})=\tfrac{D}{R}(\sum_{d_k\in\Gamma} b_k r_k+\sum_{d_k\notin\Gamma} b_k r_{Ik}^{[*]})+\lambda(\sum_{d_k\in\Gamma}\tfrac{l_k}{r_k}+\sum_{d_k\notin\Gamma}\tfrac{l_k}{r_{Ik}^{[*]}}-D) \tag{13}$$

Let $\frac{\partial F_3(r)}{\partial r_k}$=$\frac{D}{R}b_k$+$\lambda\left(\frac{-l_k}{r_k^2}\right)$=0 ($\forall d_k\notin\Gamma$), we can derive Equation (14).

$$r_j : r_k = \sqrt{\frac{l_j}{b_j}} : \sqrt{\frac{l_k}{b_k}}, \forall d_j, d_k\in\Gamma \tag{14}$$

By combining Equation (12), Equation (14), and $T^{[*]}=D$, we can finally derive Equation (11). ■

Theorem 2: **Given** a task t_i with a predefined deadline $D(t_i)$, a candidate node p_s with bounded available resource vector $\boldsymbol{a}(p_s)$ and price vector $\boldsymbol{b}(p_s)$, and a skewed workload vector $\boldsymbol{l}'(t_i)$ subject to Inequality (5), **then** t_i's payment under resource allocation $\boldsymbol{r}_E^*$ conforms to Inequality (15), where η=$\sum_{i=1}^{|\Omega|}\frac{l_k}{a_k}$, μ=$\sum_{i=1}^{|\Omega|} b_i a_i$, and Ω denotes the set of resource dimensions accumulated by Line 5 of Algorithm 1.

$$P_E^* \leq \frac{1}{\alpha}\cdot\frac{D}{\frac{D}{\beta}-\theta\eta}P_I^{[*]} + \frac{\mu D}{\alpha R} \tag{15}$$

Proof: Let Ω denote the resource dimension set accumulated by the line 5 of Algorithm 1, that is, $\forall d_i\in\Omega$, $r_{Ei}^* = a_i$. We denote $r_I^{[*]}(\Omega)$ (abbreviated as $r_I^{[*]}$) the optimal resource allocation calculated in terms of Definition 3, subject to the condition that $\forall d_i\in\Omega\ r_i = r_i^*$. $T_I^{[*]}$ (short for $T_I^{[*]}(\Omega)$) and $P_I^{[*]}$ (short for $P_I^{[*]}(\Omega)$) denote the corresponding execution time and payment cost respectively.

We can write P_E^* and $P_I^{[*]}$ as Equation (16) and Equation (17) respectively.

$$P_E^* = \frac{1}{R}\cdot T_E^*\cdot(\sum\nolimits_{i=1}^{|\Omega|} b_i a_i + \sum\nolimits_{i=1}^{|\Omega|} b_i r_{Ei}^*) \tag{16}$$

$$P_I^{[*]} = \frac{1}{R}\cdot T_I^{[*]}\cdot(\sum\nolimits_{i=1}^{|\Omega|} b_i r_{Ii}^* + \sum\nolimits_{i=|\Omega|+1}^{R} b_i r_{Ii}^{[*]}) \tag{17}$$

Via Equation (11), we can get the following derivations.

$$\begin{aligned} &R\tfrac{P_E^*}{T_E^*} - \sum_{i=1}^{|\Omega|} b_i a_i = \sum_{i=|\Omega|+1}^{R} b_i r_{Ei}^* \\ &= \sum_{k=|\Omega|+1}^{R}\tfrac{b_k\cdot\theta}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_i}{a_i}}\left(\sum_{i=|\Omega|+1}^{R}\sqrt{b_i l'_i}\right)\sqrt{\tfrac{l'_k}{b_k}} \\ &= \tfrac{\theta}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_k}{a_k}}\left(\sum_{i=|\Omega|+1}^{R}\sqrt{b_i l'_i}\right)^2 \end{aligned} \tag{18}$$

$$R\frac{P_I^{[*]}}{D}-\sum_{i=1}^{|\Omega|}b_i r_{Ii}^{*}=\sum_{i=|\Omega|+1}^{R}b_i r_{Ii}^{[*]}$$
$$=\sum_{k=|\Omega|+1}^{R}\frac{b_k\cdot\theta}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l_i}{r_{Ii}^{*}}}\left(\sum_{i=|\Omega|+1}^{R}\sqrt{b_i l_i}\right)\sqrt{\frac{l_k}{b_k}} \quad (19)$$
$$=\frac{\theta}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l_i}{r_{Ii}^{*}}}\left(\sum_{i=|\Omega|+1}^{R}\sqrt{b_i l_i}\right)^2$$

We can derive Ineq. (20) based on the above equations.

$$\frac{R\frac{P_E^{*}}{T_E^{*}}-\sum_{i=1}^{|\Omega|}b_i a_i}{R\frac{P_I^{[*]}}{D}-\sum_{i=1}^{|\Omega|}b_i r_{Ii}^{*}}\le\frac{D-\theta\sum_{i=1}^{|\Omega|}\frac{l_i}{r_{Ii}^{*}}}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_k}{a_k}}\beta \quad (20)$$

Finally, based on Inequality (20), $\sum_{i=1}^{|\Omega|}\frac{l_i}{r_{Ii}^{*}}>0$ and Inequality (5), we can derive the conclusion as follows:

$$P_E^{*}\le\frac{1}{R}\cdot T_E^{*}\cdot\left(\sum_{i=1}^{|\Omega|}b_i a_i+\beta\frac{D-\theta\sum_{i=1}^{|\Omega|}\frac{l_i}{r_{Ii}^{*}}}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_k}{a_k}}\cdot(R\frac{P_I^{[*]}}{D}-\sum_{i=1}^{|\Omega|}b_i r_{Ii}^{*})\right)$$
$$\le\frac{D}{\alpha R}\cdot\left(\sum_{i=1}^{|\Omega|}b_i a_i+\beta\frac{D-0}{D-\theta\sum_{i=1}^{|\Omega|}\frac{\beta l_k}{a_k}}\cdot(R\frac{P_I^{[*]}}{D}-0)\right)$$
$$=\frac{D}{\alpha R}\cdot\left(\sum_{i=1}^{|\Omega|}b_i a_i+\frac{\beta R}{D-\theta\sum_{i=1}^{|\Omega|}\frac{\beta l_k}{a_k}}P_I^{[*]}\right)$$
$$=\frac{1}{\alpha}\cdot\frac{D}{\frac{D}{\beta}-\theta\sum_{i=1}^{|\Omega|}\frac{l_k}{a_k}}P_I^{[*]}+\frac{D}{\alpha R}\sum_{i=1}^{|\Omega|}b_i a_i$$
$$=\frac{1}{\alpha}\cdot\frac{D}{\frac{D}{\beta}-\theta\eta}P_I^{[*]}+\frac{\mu D}{\alpha R}$$ ■

V. Experimental Analysis of Payment Bound

In this section, we first extend the basic optimal algorithm (Algorithm 1) to adapt to a dynamic situation, by taking into account the dynamic changes of resource states and possibly erroneous workload prediction. Then, we evaluate such an improved algorithm on a close-to-real cloud prototype deployed with 56 virtual machines, and intensively analyze the experimental results about payment bound.

A. Dynamic Min-Payment-OPT Algorithm

In the dynamic situation, we allow each task to own a series of inter-dependent subtasks, and the succeeding subtasks cannot be started unless its preceding subtask is finished. Each subtask could be considered a web service (a.k.a., service) developed and deployed by a third part (say another developer). The execution efficiency and the payment of each task are closely related to the availability of the hosts on which its subtasks would run and the prices that their owners assigned. In addition, since there are strict dependency relation among the subtasks, we extend our algorithm to dynamically make the resource allocation decision for any subtask whenever its preceding one is just finished, such that each allocation could be with more accurate real-time information. In order to conform to our previous problem formulation, each subtask is assumed to be a CPU-intensive service thus θ will be set to 1 here. The pseudo-code of the extended dynamic version of our optimal algorithm is shown below.

Algorithm 2 Dynamic Min-Payment-OPT Algorithm

Input: deadline $D(t_i)$, t_i's subtask set $\Pi(t_i)$;
Output: Dynamically execute $\Pi(t_i)$ with minimized payment cost;
1: $D'=\alpha\cdot D(t_i)$; /*Use a tuned deadline*/
2: **for** (each subtask $st_j\in\Pi(t_i)$ in order, j=1,2,$\cdots$,$|\Pi(t_i)|$) **do**
3: $\Gamma=\Pi(t_i)$, $C=D'$, $\boldsymbol{r}^{*}=\varnothing$ (empty set);
4: Retrieve current resource states $\boldsymbol{a}=\{a(p_1),a(p_2),\cdots,a(p_n)\}$;
5: $\boldsymbol{r}_{\Gamma}^{(*)}$ = Co-Step(Γ,C); /*Compute optimal r based on Γ*/
6: **if** ($r_{\Gamma_j}^{(*)}>\max(\boldsymbol{a})$) **then**
7: $r_{\Gamma j}^{*}=\max(\boldsymbol{a})$; /*$r_{\Gamma_j}^{*}$ denotes optimal allocation of st_j*/
8: **else**
9: $r_{\Gamma j}^{*}=r_{\Gamma j}^{(*)}$;
10: **end if**
11: Select execution node p_e such that $a(p_e)\ge r_{\Gamma_j}^{(*)}$ and no other node p_o satisfies $r_{\Gamma_j}^{(*)}\le a(p_o)<a(p_e)$; /*to maintain system availability.*/
12: $\Gamma=\Gamma\backslash st_j$; /*$\Gamma$ takes away st_j*/
13: $C=C-T(st_j)$; /*Update C by cutting st_j's time ($\approx\frac{l(st_j)}{r_{\Gamma_j}^{*}}$).*/
14: Start executing st_j on p_e;
15: Sleep until st_j is completed;
16: **end for**

At the beginning of the algorithm, the deadline will be set to D'=α·$D(t_i)$ before performing the Co-Step operation, where α is the lower bound of the estimation ratio according to Inequality (5). After that, the subtasks of the user task will be executed one by one until all of them are completed. Everytime a subtask is finished, its succeeding task will be triggered and our algorithm will allocate the most appropriate resource share to it by performing the Co-Step operation. $\max(\boldsymbol{a})$ denotes the maximum value among the elements of the set $\boldsymbol{a}$. When the resource share to be allocated is determined (through line 5 $\sim$ 10), the qualified node whose availability is closest to the resource demand will be selected as the execution node (line 11), for keeping high availability for the tasks with large resource demand.

Compared to Algorithm 1, the key changes of Algorithm 2 is the value of the deadline used (line 1) and when to determine the resource allocated to the subtasks to be executed (line 6$\sim$15). With a little stricter deadline, the Algorithm 2 could strictly confine the task's execution time before its deadline even though the task's execution property were predicted imprecisely. By dynamically determining the resource share for any subtask only if its previous subtasks are all completed, the adaptability to the changes of resource states will get much higher in Algorithm 2. In addition to these two features, we could further improve the robustness of the algorithm by introducing a queueing policy. Note that for Algorithm 2, once all physical resources are already allocated, the newly arrival subtasks cannot be executed immediately until more resources are released. In other words, they will be failed if there are no any queuing policy

to buffer/queue the subtasks temporally. On this point, we just need to make a minor revision to Algorithm 2: before performing line 5, if $\boldsymbol{a}$ is empty or $\max(\boldsymbol{a})$ is smaller than the subtask's least requirement, the subtask will be queued temporally to avoid direct failure of the task. In the next section, we will evaluate the Algorithm 2 with such a queuing policy in a real cluster environment. On the other hand, if a subtask is allocated with a suboptimal CPU capacity (i.e., $r_i^* < r_i^{(*)}$) due to the limited available resource states when it was scheduled, it is viable to further improve the resource allocation (say further minimizing the payment cost) by dynamically increasing its allocated resources based on the newly released resources over time. To implement this function, a new observer thread is needed to periodically check all hosts' real-time states and all running subtasks' allocated resource will be tuned from time to time to reach their theoretical optimal amount as closely as possible.

B. Experimental Setting

We implement a composite cloud service prototype that can help solving any dense-matrix problems. Matrix computation serves as such a fundamental domain in mathematics, that quite a few Grid services [19], cloud services [20], and web services [21] have been developed to suit ease-of-use demands for researchers on linear-algebras.

We implement a cloud service prototype that can help solving any complex matrix problem that contains a series of nested matrix computations. In our experiment, we use 8 physical nodes behind a cluster, and each node owns 2 quad-core Xeon CPU E5540 (i.e. 8 processors per node) and 16GB memory size. There are 56 VM-images (centos 5.2) maintained by Network File System (NFS), so 56 VMs (7 VMs per node) were generated at the bootstrap. XEN 4.0 [15] serves as the hypervisor on each node and dynamically allocates various CPU rates to the VMs at run-time Through XEN's credit scheduler [22].

We make use of *ParallelColt* [23] to perform the matrix computation. ParallelColt [23] is such a library that can effectively calculate complex matrix-operations like matrix-matrix multiply, in parallel via multiple threads. In our benchmark, we simulate a large number of user requests, each of which is composed of 3$\sim$15 subtasks. Each subtask is constructed by one of three typical matrix-operations (i.e., matrix-multiply, matrix-power, and QR-matrix-solving(least-square)) with various parameters assigned. That is, each request contains many subtasks that are randomly selected from the above three types. We evaluate our algorithm under different competitive situations with various number (1$\sim$40) of tasks submitted simultaneously, thus there are 40 cases for each experiment and 820 tasks in total as observed.

Each user request (denoted as task t_i) is assigned with a deadline, a random value in $[\frac{1}{8}\cdot T_1(t_i), T_1(t_i)]$, where $T_1(t_i)$ means the estimated execution time when running t_i on one core. Based on our experiment, the three matrix operations on one core cost from 1 second to 1206 seconds, which means a heterogenous workload state. In addition, we observe that the lower bound of the workload predicted is always 0.7 times as high as the real workload calculated after its execution. Hence, in the experiment, the value of α will be set to 0.7 or 1, to evaluate the performance between with and without a tuned deadline.

C. Experimental Results

We first characterize the Payment Extension Ratio (PER), when using the user-expected deadline in the dynamic Min-Payment-OPT algorithm. That is, in the Algorithm 2, α will be set to 1 at the first line (i.e., D'=$D(t_i)$). Figure 1 shows the PER values in the 40 cases with 1 - 40 tasks submitted respectively. We can clearly observe that in the situation with relatively low resource competition (e.g., with less than 15 tasks submitted), the maximum values of PER always stay around 1.1 times as large as their theoretically optimal values. With increasing number of competitive tasks submitted, the PER's maximum values will increase accordingly, up to about 1.25 in most of cases. The mean value of PER is always kept around 1.05, regardless of the resource competition status. This means that a large majority of task payments are already minimized to be close to their theoretically optimal values.

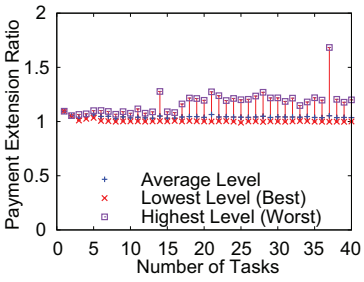

Figure 1. Payment Extension Ratio (α=1, i.e., D'=D)

Figure 2 presents the experimental results when using the tuned deadline D'=αD (where α=0.7) in our Min-Payment-OPT algorithm. The stricter-deadline based algorithm can guarantee task execution length be within its user-expected deadline, even though the task workload was predicted with a certain error as defined in Definition 1. From Figure 2, we observe that when there are less than 30 tasks submitted, the maximum and mean PER is always lower than that when α is set to 1 (as shown in Figure 1). However, when there are over-many tasks submitted (e.g., about 35 tasks), the maximum and mean values of PER observed will increase significantly. This is because the user payment is proportional to task execution length as defined in Equation (2), while some tasks' execution lengths will be extended inevitably because of the increasing competition level. In addition, the average value of PER is always very close to 1, since about 95% of tasks' PERs are smaller than 1.15 as observed, which means majority of task payments will still be satisfied by users with a high probability.

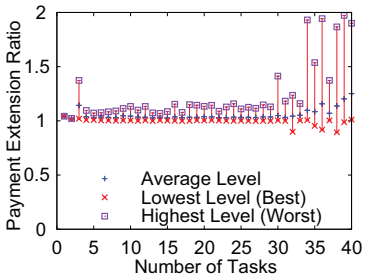

Figure 2. Payment Extension Ratio (α=0.7, i.e., D'=αD)

We present the distribution of PER under the two algorithms (with D'=D and D'=αD respectively) in Figure 3. Through Figure 3, we can clearly observe that the original-deadline-based algorithm leads to lower payment cost for majority of tasks. That is, the original-deadline-based algorithm shows prominently higher adaptability to the resource competition level. Such observations constitute a strong foundation that may lead us to combine the two algorithms together to maximize the system-wide performance at various situations with different levels of competitions, which could be our future work.

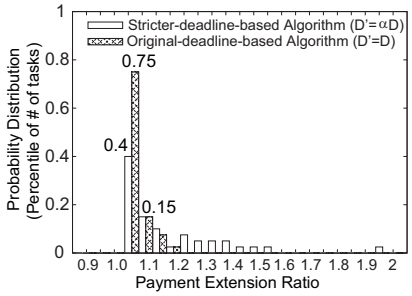

Figure 3. Distribution of PER

Finally, we evaluate the stability (or fairness index) of the task payment among all tasks. Based on Jain's work [24], fairness index (higher value means higher fairness) is defined as Equation (21) whose value ranges in [0,1], where φ_i refers to the PER of task t_i.

$$F(\varphi) = \frac{\left(\sum_{i=1}^{n} \varphi_i\right)^2}{n \sum_{i=1}^{n} \varphi_i^2} \tag{21}$$

As observed in Figure 4, the fairness index of PER is always kept over 0.99 for both cases under the relatively uncompetitive situation (e.g., $m \leq 30$), and still kept about 0.95 in the case with higher competition (i.e., when $m > 30$). Recall that there are only 10 physical machines used for resource provisioning in our experiment, this means that the user payments under our Min-Payment-OPT algorithm can be kept very stable with such a dense resource consolidation.

VI. Related Work

Recently, there are many existing works (such as [12], [7], [8], [9], [3], [25] studying the optimization problem of resource allocation on the cloud computing model. Whereas, none of them studied the payment bound as deeply as in this paper, from both theoretical and experimental perspectives.

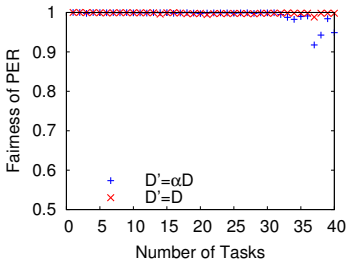

Figure 4. Fairness Index of PER

Most of cloud systems adopt similar models, aiming to optimize task execution efficiency by leveraging VM technology and minimize user payment meanwhile. Li et al. [12] proposed two online greedy scheduling methods, namely dynamic cloud list scheduling (DCLS) and dynamic cloud min-min scheduling (AMMS), to minimize the mean task execution length. Similarly, Salehi et al. [7] proposed another dynamic greedy algorithm that repeatedly tune resource allocation at run-time based on updated information.

Artur et al. [25] propose a probabilistic model for the optimization of monetary costs, performance, and reliability, given user and application requirements and dynamic conditions. Their research is based on a real instance price traces provided by Amazon EC2 [2].

In addition to the above works that are based on experiences, the cloud research performed by Mao and Marty seems more interesting. They formulated the cloud scheduling issue as an auto-scaling problem [8], [9]. That is, their design can automatically scale the resource capacity up or down according to the conditions users define. They considered both batch-mode jobs (embarrassing parallel job) and workflow-mode jobs respectively. However, their research is based on a coarse knowledge about resource (such as the number of VM instances) and task workload (such as the number of jobs/tasks), which will significantly limit the optimality of their solution. Moreover, they did not provide any bound analysis (e.g., the upper bound of payment at the worst case) as our work.

Our previous work [3] is another related work to this paper. In that work, we modeled the cloud resource allocation issue to be a convex optimization problem, by leveraging the divisible resource feature due to VM technology. A provable optimal solution (called Min-Payment-OPT) is proposed, and the upper bound of task execution length is theoretically analyzed as compared to the user-specified deadline. Compared to the previous work, we make some new contributions in this paper: (1) we further analyze the payment bound under such a cloud model, by assuming that task's workload were predicted inaccurately; (2) we extend the basic Min-Payment-OPT algorithm to be a dynamic version, that can adapt to the run-time resource state changes; (3) We finally evaluate the payment bound through a close-to-real cloud service prototype built on top of a cluster environment with 56 virtual machines.

VII. Conclusion and Future Work

In this paper, we comprehensively analyzed the bound of user payment, for a payment minimization resource allocation algorithm in cloud systems. The research contribution is two-fold. (1) We theoretically derive the upper bound of user payment, in the situation where the task workload were predicted with errors. To our best knowledge, this is the first attempt in analyzing the payment bound in the context of cloud computing. (2) We rigorously evaluate the bounds of practical payment through a close-to-real cloud prototype, deployed with 56 VMs and XEN 4.0 hypervisor. Experiments confirm that our payment-minimization algorithm can keep 95% of users' payments below 1.15 times as large as the theoretically optimal values, and the payment extension ratio for the rest user payments is about 1.5 at the worst case. At present, we are completing ease-of-use interfaces based on our prototype. In the future work, we will study how to tolerate task/node failure events with checkpoint/restart mechanism based our theories on resource allocation.

Acknowledgments

This work was made by the ANR project Clouds@home (ANR-09-JCJC-0056-01), and also in part by a Hong Kong UGC Special Equipment Grant (SEG HKU09).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] Amazon elastic compute cloud: on line at http://aws.amazon.com/ec2/.

[3] S. Di and C.-L. Wang, "Error-tolerant resource allocation and payment minimization for cloud system," in *Transactions on Parallel and Distributed Systems (TPDS)*, Special Issue, 2012.

[4] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2009.

[5] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services." in *10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10)*, 2010, pp. 13–31.

[6] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," in *IEEE International Conference on Cloud Computing (CloudCom'10)*, pp. 418–425, 2010.

[7] M. A. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing," in *ICA3PP (1)*, 2010, pp. 351–362.

[8] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID2010)*, 2010, pp. 41–48.

[9] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, New York, NY, USA: ACM, 2011, pp. 49:1–49:12.

[10] H. Goudarzi, M. Ghasemazar, and M. Pedram, "Sla-based optimization of power and migration cost in cloud computing," in *The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*, 2012.

[11] L. S. Bird and J. B. Smith, "Pacora: Performance aware convex optimization for resource allocation," in *in HotPar '11: 3rd USENIX Workshop on Hot Topics in Parallelisation (Poster Session)*, 2011.

[12] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel Distributed Computing*, pp. 666–677, 2012.

[13] S. Di, C.-L. Wang, L. Cheng, and L. Chen, "Social-optimized win-win resource allocation for self-organizing cloud," in *International Conference on Cloud and Service Computing (CSC'11)*, 2011, pp. 251–258.

[14] L. Huang, J. Jia, B. Yu, B.G. Chun, P. Maniatis, and M. Naik, "Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression," in *Proceedings of 24th International Conference on Neural Information Processing Systems (NIPS'10)*. 2010, pp. 1–9.

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP'03)*, New York, NY, USA: ACM, 2003, pp. 164–177.

[16] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[17] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence windows," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 7, pp. 925 –938, july 2010.

[18] Q. Zhang, , J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in google's compute clusters," in *Large Scale Distributed Systems and Middleware Workshop (LADIS'11)*, 2011.

[19] E. Agullo, C. Coti, J. Dongarra, T. Herault, and J. Langou, "QR Factorization of Tall and Skinny Matrices in a Grid Computing Environment," in *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, 2010, pp. 1–11.

[20] Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, and Z. Zhang, "MadLINQ: large-scale distributed matrix computation for the cloud", in *Proceedings of the 7th ACM european conference on Computer Systems (EuroSys'12)*, 2012, pp. 197–210.

[21] P. Benner, R. Mayo, E. Quintana-Ortł, G. Quintana-Ortł, "Enhanced Services for Remote Model Reduction of Large-Scale Dense Linear Systems", in *Applied Parallel Computing*, 2006, pp. 329–338.

[22] Xen-credit-scheduler: http://wiki.xensource.com/xenwiki/creditscheduler.

[23] P. Wendykier and J. G. Nagy, "Parallel colt: A high-performance java library for scientific computing and image processing," *ACM Trans. Math. Softw.*, vol. 37, pp. 31:1–31:22, September 2010.

[24] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*, John Wiley & Sons, April 1991.

[25] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10)*, Washington, DC, USA, 2010, pp. 257–266.